N92-14670 51-63 48235 P 33

AUTOMATED IMPLEMENTATION OF RULE-BASED EXPERT SYSTEMS WITH NEURAL NETWORKS FOR TIME-CRITICAL APPLICATIONS

P.A. Ramamoorthy, Song Huang and Girish Govind

Department of Electrical & Computer Engineering

University of Cincinnati, M.L. #30

Cincinnati, Ohio 45221-0030

ABSTRACT

In fault diagnosis, control and real-time monitoring, both timing and accuracy are critical for operators or machines to reach proper solutions or appropriate actions. Expert systems are becoming more popular in the manufacturing community for dealing with such problems. In recent years, neural networks have revived and their applications have spread to many areas of science and engineering. A method of using neural networks to implement rule-based expert systems for time-critical applications is discussed in this paper. This method can convert a given rule-based system into a neural network with fixed weights and thresholds. The rules governing the translation are presented along with some examples. We also present the results of automated machine implementation of such networks from the given rule-base. This significantly simplifies the translation process to neural network expert systems from conventional rule-based systems. Results comparing the

performance of proposed approach based on neural networks vs. the classical approach are given. In this paper, the possibility of VLSI realization of such neural network expert systems is also discussed.

1. INTRODUCTION

1.1. Expert Systems

In recent years, more and more automated manufacturing systems are being developed. Expert systems are playing important rules in these highly complex systems for decision-making, real-time operational control, fault diagnosis and so on (Heragu and Kusiak, 1987). Expert systems are computer programs that utilize a significant amount of expert knowledge about a particular domain to solve problems in that domain. For example, they are being considered or applied to problems such as machinery monitoring, part design, robot control, process control, fault diagnosis and so on. Such expert systems are highly appealing for real-time monitoring of dynamic systems as an alternative to human supervision. The response of human operators in such time-critical applications may vary considerably leading perhaps to omission of relevant information, inconsistent responses and even panic. The expert systems can respond consistently and will also be able to arrive at proper decisions at a much fast rate. Other attributes of expert systems are expendability, flexibility, provision for explanation and ability to work on incomplete or inexact information (Harmon, Maus and Morrissey, 1988).

A fully established expert system contains four major parts (Hayes-Roth, Waterman and Lenat, 1983), they are knowledge base, inference engine, knowledge acquisition, and explanatory interface. The knowledge base and the inference engine are the core of an expert system. For real-time systems, the explanatory interface with human operators may not be needed because of the time overhead involved. And the knowledge acquisition interface could be a hardware module between the expert system and the dynamic system to be monitored. Figure 1 shows the diagram of a typical expert system (Forsyth, 1984). A knowledge base contains facts and rules. Facts are short-term information that can change rapidly, and rules are long-term information about how to generate new facts or hypotheses from what are already in the knowledge base.

Most expert systems employ an IF-THEN format to represent rules in the knowledge base. The IF-THEN rules are of the following form:

IF (antecedents)
THEN (consequent).

Rules are conditional statements i.e. if the antecedents are satisfied then the actions defined by the consequent part of the rules should take place (Hayes-Roth, 1985). Two rules may fire at the same time if their antecedents are satisfied at the same time provide that one's consequent is not in the other's antecedents. This property invites the possibility of parallel implementation of such rules given a proper architecture. However, most expert systems implement rules in a sequential manner, rules fire one by

one.

The function of an inference engine in an expert system is to control the selection of relevant knowledge for a given set of information. It takes rules and facts from the knowledge base as data and finds an appropriate solution. There are several inference or control strategies being developed for inference engines. The most used strategies are, forward chaining which involves reasoning from facts to goals, backward chaining which propagates from goals trying to reach the facts, and hybrid chaining which combines the previous two strategies and propagates from both sides. The design of an inference engine is very crucial since it can affect the performance of an expert system significantly.

There are two distinct phases in applying expert system concepts to a problem. The first is the development of an expert system as four components as indicated before. The second phase is the actual implementation of the expert system on a computer. The number of rules required for a useful expert system can be very large. Thus leads to an increase in the memory requirements as well as a deterioration in its speed performance. In addition, most expert systems are written on high-level interpretive languages, such as LISP and PROLOG, thus requiring fairly expensive computers to run on, and the implementation is sequential in nature. That is, given a set of facts, the inference engine selects a rule for examination for firing, and if the rule is fired, the resulting actions or conclusion is inserted into the knowledge base. Then the inference engine proceeds to the next rule and so on until all the

rules in the knowledge base are examined.

The implementation as explained above has serious consequences on the speed performance of an expert system. As the number of rules in an expert system increases, the time required to cycle through the rules as many times as may be dictated by the facts, can increase enormously making the expert system unsuitable for time-critical applications. For time-critical applications, there is a need for reaching proper solutions or decisions within a small fixed amount of time. In these situations, speed and intelligence are two primary factors for operators or machines to make right decisions or to perform proper actions. In our proposed system, both intelligence and timing are considered. In fact, our method combines the knowledge base and inference engine of expert systems together thus significantly reducing the overhead processing within the expert system.

1.2. Neural Networks

The architecture of neural networks is based on our present understanding of our brain's biological nervous systems and presumably achieve good performance via dense interconnection of simple computational elements (Lippman, 1987). Since these networks have characteristics analogous to human intelligence, using them as a tool to implement expert systems seems quite logical, because after all, expert systems are computer realizations of human intelligence over a particular domain of expertise. In addition, their massive parallelism and relatively simpler architecture are advantages in implementing time-critical expert systems.

Figure 2 gives a single element of the neural network, the element is also called a neuron. A set of inputs labeled $x_1, x_2, \ldots, x_n$ are network inputs which are connected to the neuron through a set of associated weights $w_1, w_2, \ldots, w_n$. Inputs correspond to the facts as incoming signals and weights represent the strengths of the importance of inputs. The outgoing signal y is obtained as

$$y = f\left(\sum_{i=1}^{n} x_i w_i - T \right) .$$

Where T is called offset or threshold of the neuron, and f is some nonlinear function. A special case of this element is a threshold logic gate (T-gate), also called perceptron, in which all inputs and outputs are limited to binary numbers and the output function f is a binary function too. Figure 3 shows the diagram of a T-gate. The output function of a T-gate is defined as

$$y = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} x_i w_i < T \\ 1 & \text{otherwise} \end{cases} .$$

Figure 4 shows a one-layer neural network with n neurons. Each input x_i is connected to every neuron y_j and the weight between them is w_{ij}. The output function is given as the following

$$y_j = f\left(\sum_{i=1}^{n} x_i w_{ij} - T_j \right) .$$

Where T_j is the threshold value of the jth neuron in the net. Our proposed expert system is a multi-layer network like the one in Figure 5. This kind of networks are formed by cascading two or

more layers of neurons. The multi-layer neural network can accomplish some more dedicated tasks one-layer network might not be able to do. However, the algorithms for systematic training of multi-layer networks may be more complex. In our network, weights associated to all neurons are fixed binary values which are directly obtained from the given rules. The thresholds are also determined by these rules, they are binary too. Therefore we omitted the painstaking network training procedure. This feature of neural network design is unique, since we are dealing with the real-time situations and the knowledge base is already known.

In the next section, we describe how to construct a neural network from a given rule-based expert system. And in section three, the explanation of the automated translation -- an interpreter is given. Then examples to test the proposed system are presented along with the performance comparisons between our system and conventional expert systems. The conclusion section includes the summary and the possibility to realize the proposed expert system by hardware using VLSI technology.

2. CONSTRUCTING NEURAL NETWORK EXPERT SYSTEMS

2.1. Rule-based Systems and Classification

As stated before, most expert systems are using IF-THEN rules to represent their knowledge. The conditional statements of the antecedents and consequent contain some relational operators to connect several facts together. These operators are AND, OR, and

NOT, all other binary relations can be represented by these three.

The classification expert systems are very useful for real-time applications such as diagnosis, control, maintenance and process monitoring. In a classification system, most or maybe all inputs are questions which can be answered by either YES or NO (TRUE or FALSE). This feature of classification expert systems is similar to that of a threshold logic gate in which all inputs and output are binary values only, and a "1" can used to represent YES or TRUE, while a "0" for NO or FALSE. However, classification systems may not be limited to binary problems only. Any problems, discrete or continuous, can be approximated by such a model by proper coding of the inputs and outputs (Gallant, 1988). For example, we can divide continuous variable 'age' into a number of binary variables, INFANT, CHILD, YOUTH, MIDAGED, and SENIOR. Each of these variables corresponds to certain range of age:

IF (age $\leq$ 1) THEN INFANT is TRUE;

IF (age > 1 AND age $\leq$ 12) THEN CHILD is TRUE;

IF (age > 12 AND age $\leq$ 30) THEN YOUTH is TRUE;

IF (age > 30 AND age $\leq$ 60) THEN MIDAGED is TRUE;

IF (age > 60) THEN SENIOR is TRUE.

Therefore, age=16 would cause YOUTH to be TRUE, or YOUTH=1. Any continuous variables can be approximated by classification systems to arbitrary precision using this coding technique theoretically. Therefore restricting ourself to classification expert systems dose not limit our abilities for solving continuous or complex problems.

2.2. AND/OR Inference Net

An AND/OR tree is an equivalent way to show what a set of antecedent-consequent rules can do (Winston, 1984). In which, the consequent of one rule is connected to other rules as antecedents. Figure 6 is an example of such a tree. The rectangular nodes represent output assertions or consequent of rules they associate, and the circular nodes are for input data or intermediate results, or antecedents of rules. If there is an arc appearing to join several or all descendent nodes of a higher-level node, then this node is called an AND node. Nodes without arcs to connect their descents are OR nodes. Any collection of IF-THEN rules defines such a tree. Therefore, for any given expert system, if the collection of IF-THEN rules are known, a AND/OR inference tree(net) could be always constructed which supplies a graphic view for connections of its elements. For the design of time-critical neural network expert systems, we assume that the rule-bases of the systems are available. Thus there exists an AND/OR inference tree for each expert system. Our main goal is to convert this kind of inference network of a given rule-base into a neural network.

For large complicated expert systems, the number of intermediate results can be large. These intermediate results may be used to produce other intermediate outputs or final outputs, or for just explanatory purposes. In some expert systems, there are a lot of intermediate outputs to serve as tools to break the large rules into several small ones, and some of them are merely for the purpose of conjunction of rules. If we can cut the number of these

intermediate results nodes to the minimum, then there would be a big gain in speed and efficiency. In our design of neural network expert systems, we make the rearrangement of the connections, inputs and outputs to minimize the intermediate results or even eliminate some dummy rules served as conjunctions of other rules.

2.3. Weights and Threshold of AND/OR Gates

From the discussion of previous section, the whole expert system's rule-base can be decomposed into AND connections and OR connections. Then the work of constructing a neural network type expert system reduces to transforming the AND/OR graph of the expert system into networks composed of perceptrons. As indicated above, the inputs and outputs of the system are either binary values or recoded into such values. Thus the remaining task is to find the number of types of perceptrons needed. Generally, we need only two types of perceptrons, one for AND operations, and the other for OR operations. We can call them AND T-gate and OR T-gate. The mixed operations of AND/OR can be represented by the combinations of these T-gates.

The procedures of assigning weights and threshold values to different T-gates are given as the following. We may also introduce negative weights into the T-gates for the purposes of performing negation operations. But for the sake of simplicity, we limit the weights to binary values. If a negation is needed for a input data of the system, we can create a new input data for it. For example, if "moving" is an input data of a robotics control rule-base, and its negation"not moving" is also needed in the input fact set, we

then can set two input values, MOVE for "moving", and NMOVE for "not moving".

2.3.1. AND T-gate

For AND operations, the weights of corresponding T-gate are either one or zero. Weight "one" is assigned to inputs which take part in the AND operations, while "zero" to others. The threshold value of the AND T-gate equals the number of inputs take part in the AND operation. Figure 7 is an AND T-gate for the following rule:

IF A and B and C and E

THEN X .

Here, A, B, C, D, and E are inputs, and X is the output. Since inputs A, B, C, and E are in the AND operations of the rule, therefore they are assigned with weight "one". Input D does not take part in the AND operation, so its weight is "zero". Usually we do not explicitly put zero inputs into the graph just like we do not put unnecessary inputs into rules. But for neural networks, all the inputs are connected into neurons no matter they are needed inputs or not. The threshold value of this AND T-gate is four, because there are four inputs in the AND operations. The output X of this T-gate is one if A, B, C, and E are all one. Otherwise, X is zero.

2.3.2. OR T-gate

For OR operations, the weights assignment is the same as for the AND T-gate. The threshold value of the OR T-gate is always one. This means that the output value of the OR T-gate will be one if

any of its inputs is one. Zero weights are assigned to those inputs which do not take part in the OR operations. Figure 8 is an OR T-gate for the following rule:

IF S or T or U or W

THEN Y .

Here, S, T, U, V, and W are inputs, and Y is the output. Since inputs S, T, U, and W are in the OR operations of the rule, therefore they are assigned with weight "one". Input V does not take part in the OR operation, so its weight is "zero". The output Y of this rule is one if any of S, T, U, or W is one, otherwise, Y is zero.

2.3.3. AND/OR T-gate

For a small number of mixture of AND with OR operations, a one layer T-gate is possible to simulate these operations. But generally, if the antecedent logic expression contains both AND operations and OR operations, then the separation of ANDs and Ors is necessary. We use a two-layer network to represent the corresponding rule. The first layer contains several AND T-gates (or T-gates, this depends on the given rule), and the second layer only has one OR T-gate (or one AND gate). Figure 9 shows the mixed T-gate network for the following rule:

IF A and B or C and D

THEN X .

Here, in the first layer, there are two AND T-gates. One for the operation "A and B", and one for "C and D". The weights and thresholds of these two AND T-gates are assigned according to the

rules given in section 2.3.1. The outputs of these AND T-gates are the inputs of the second layer OR T-gate. The weights and the threshold value of this OR gate are all "one" according to the rules in section 2.3.2.

As stated above, some simple mixed rules can be converted to one layer T-gate network. For example, rule:

IF A and C or A and D

THEN X .

The antecedents can be written in another form: "A and (C or D)". Then we will be able to use one T-gate to simulate this rule as in Figure 10.

The weights of these one layer T-gate networks are not binary anymore.

3. STRUCTURE OF A MACHINE TRANSLATOR

Since the weight and threshold assignment is very regular, we can automate the process of obtaining the perceptron based neural network form any rule-based classification expert systems. The weights and thresholds of neurons in these networks are obtained directly from the given rule-base. Therefore, there is no painstaking training procedure for determining weights and thresholds for the networks. The translation is an off-line process, so the time used on translation will not affect the speed performance of the system.

Figure 11 is a general diagram which shows the required

translation steps the designed automated translator needs to accomplish. The precode process is not included for continuous data or for data have multiple values. The continuous or multi-valued data can be converted to binary data by the methods introduced in section 2.1.

This automated machine translator is divided into two major parts, a compiler which reads in the conventional rule-based knowledge system and separates inputs, outputs and the rules, and a converter which reads in the intermediate results from the compiler and constructs a neural network architecture. The process involved in the design of this machine translator is explained in this section.

3.1. Compiler

The compiler of this translator performs the operations much like a conventional compiler dose which takes source code as input and generates machine code as output. The difference here is that the source code is a set of rules and the output of this compiler are intermediate information for the use of next stage of the translation. The design modules of this compiler are given in Figure 12. The explanations of these modules are presented below.

3.1.1. Lexical Analysis

This module reads in the input file which contains the rule set of a conventional expert system. The rules in the input file are modified from their original forms. Since there are many different expert system shells or tools specifying rules, using any of those representations of rules as our format is not appropriate.

Thus, we choose the one which is most close to the normal written form for IF-THEN rules, which is given as below:

RULE n: title

IF (condition 1) op (condition 2) op ...

THEN (solution 1) (solution 2) ...

CF = d .

Where "n" is a integer used to number rules, "title" gives the name or a short message of the rule. The relational operator "op" may be AND or. The certainty factor "CF" specifies the probability of a rule to be true when it is fired, and its value "d" is a decimal number ranging from zero to one. This certainty factor is not used in our system, including this factor in it is for further study of such systems with fuzzy concepts. The following example shows that how to modify a given rule to the form which is recognized by the compiler.

Given a automobile diagnosis rule of the following:

R7: IF the battery and connecting wires are not at fault, and you turn the headlights on and try to crank the engine, and lights remain bright or dim only slightly,

THEN check the starter, solenoid and wiring.

The modified rule would be:

RULE 7: Starter, solenoid, Wiring Trouble

IF (BTY = OK) AND (BTY_CTG = OK) AND (HL = ON) AND (EGN = CKG)

AND ((LTS = BRT) OR (LTS = DIM_S))

THEN (CHK = STAT) (CHK = SLD) (CHK = WRG)

CF = 1.0 .

The translation of names and features of the original rule is easily understood.

The lexical module reads the characters in the source file and groups them into a stream of tokens in which each token represents a logically cohesive sequence of characters (Aho, Sethi and Ullman, 1986). Such tokens may be a keyword (IF, THEN, CF, etc.), an operator (AND, OR), a number, an identifier (CHK, BTY, HL, etc.) or a parentheses. This module also checks if there are any lexical mistakes, such as missing keyword, unpaired parentheses, mistypings.

3.1.2. Symbol Table

A symbol table is a data structure containing a record for each name or identifier, with fields for the attributes of the name. The symbol table is the core of our compiler which keeps track of each name and its attributes such as type, rule numbers in which this name appears, class, and size. When a name in the source rule-base is detected by the lexical analysis module, this name is entered into the symbol table and along with some of its attributes. However, all attributes of a name may not be entered at once, since it maybe appear in several rules.

One important attribute of the names is the their types. One name can have one of three different types, input, output, or intermediate. The input-type names are the input data of the system, which will also be the input data of the final neural

networks. While the output-type names are those leading to conclusions or actions taken by the system, they are the outputs of last-layer neurons of the final neural networks. The intermediate names are subject to be eliminated by the optimization module if possible, since they might be only used locally as conjunctions of two connected rules.

Another feature of the names is their class. If a name is used in an AND operation, then it will have an AND class. An OR class is defined similarly for a name. If a name is used in both AND OR operations, it will be assigned AND/OR class. This class attributes will be useful for weights assignment during the conversion part of the translation.

3.1.3. Logical Analysis

Since the compiler is mostly dealing with logical expressions, it is needed to have a phase in the program to perform logical operation analysis. During this operation, the program will classify the classes of names or symbols on the symbol table. For some complicated rules, the logical analysis module will perform logic simplification operations to them. After this process, the conditions of rules should be in either conjunctive normal form (CNF) or disjunctive normal form (DNF) (Davis and Weyuker, 1983). Two examples to show these two normal forms are given below.

CNF: (A or B) and (C or D) and (E or F)

DNF: (A and B) or (C and D) or (E and F)

From the discussions of previous sections, these two forms can be represented by corresponding T-gates.

If the conditions in a rule is very complex and one name is used for more than once in the same rule, then the break-down process might be needed to separate this rule into two or even more sub-rules. This is considered to be a setback of the design, but most rules in practical applications are simple and can be simulated by only one or two layers of T-gates.

3.1.4. Relation Analysis

Most rules are given in a sequential order. This means that the output of one rule is one of the inputs of other rules, unless this rule is the output rule itself. The compiler needs to know what orders between rules are. This module provides the information which links rules together. It also marks the first layer (input) rules and the last layer (output) rules.

Usually, rules are connected in the feedforward style. Facts or input data yield sub-goals or intermediate results, these results are fed into next level rules, and finally reach the goal or output data. In the conventional expert system, inference engine fires rules one by one. The inferencing process stops when there are no more rules to be fired. Therefore, it will not cause problem if one output is fed back to higher level rules (in the network point of view). But for a neural network, this is to be considered a feedback, connection from higher layer neuron back to lower layer neurons. One might get false results if not waiting for long enough. In our design, only one layer feedback is allowed. Thus, if the results of two runs are the same, then the system should stop.

3.1.5. Optimization

This module checks if there are rules which can be combined together. For example, we can combine

RULE 1: IF (conditions) THEN (do X),

RULE 2: IF (do X) THEN (do Y),

to form a new rule:

RULE 1.2: IF (conditions) THEN (do X) AND (do Y).

The optimization process also can be applied to variables which only served as conjunctions between rules. For following example:

RULE 5: IF (conditions) THEN (set flag1),

RULE 6: IF (flag1 is set) THEN (actions).

The variable "flag1" is used only as a conjunction between RULE 5 and RULE 6, then we can create a new rule:

RULE 5.6: IF (conditions) THEN (actions).

3.1.6. Output Function

The compiler needs to pass its analysis and symbol table as well as the encoded rules to the next part -- conversion. This module produces output files that will be the input of the conversion process.

3.2. Converter

The main purpose of this conversion process is to utilize the information obtained from the compiling phase and to build the expert system consisting of neural network elements. Therefore, the major operations in this phase are assignment of weights and calculation of thresholds for each neural element. Figure 13 is a block diagram for this conversion procedure.

The first step of this process is the rule-to-neuron

translation. For each rule in the conventional knowledge base, there should have one or more neurons to match the rule. The exception is that if the rule can be compressed by the optimization module as discussed earlier. Then for each converted rule, weight assignment should be performed according to the type of the rule.

The guidelines of how to assign weights are explained in section 2. The type of neurons are AND, OR, as well as mixed type. For a mixed type neuron, the program will separate the components in the condition clause of the rule. This separation depends on the forms of the rule. If it is in the CNF, then the first layer of the sub-net contains several OR T-gates, followed by an AND T-gate. If it is in the DNF, then the first layer composes a number of AND T-gates, followed by an OR T-gate.

Threshold values of the converted neurons are defined according to the guidelines given in section 2. The process is actually counting the number of inputs leading to the neuron, if the neuron is a AND T-gate. For an OR T-gate, the program simply assigns a one.

After all rules in the system have been converted to their neurons, the operation of connecting them into a whole network is applied. The last task of the conversion is to output the constructed neural network which will simulate the functions of the original rule-based expert system.

4. AN EXAMPLE OF A NEURAL NETWORK BASED EXPERT SYSTEM

In order to provide a clear view of the design process of the proposed neural network based expert system, we discuss a simple classification expert system example which containing fifteen rules. The rule-base of this sample system is given below (Winston, 1984).

R1: IF the animal has hair
THEN the animal is a mammal

R2: IF the animal gives milk
THEN the animal is a mammal

R3: IF the animal has feathers
THEN the animal is a bird

R4: IF the animal flies and it lays eggs
THEN it is a bird

The above four determine the biological class of mammal and bird. R1 and R2 has the OR relation, they can be combined into on rule:

R1.2: IF the animal has hair OR it gives milk
THEN the animal is a mammal.

Similarly, R3 and R4 can be compressed into:

R3.4: IF the animal has feathers OR, it flies and it lays eggs
THEN it is a bird.

Once we know that an animal is a mammal, two rules determine whether it is a carnivore.

R5: IF the animal is a mammal and it eats meat
THEN the animal is a carnivore

R6: IF the animal is a mammal and it has claws and pointed-

teeth and its eyes point forward

THEN the animal is a carnivore

Under the category carnivore, we have rules to identify specific animals, cheetah and tiger.

R7: IF the animal is a carnivore and it is brown and has dark spots

THEN the animal is a cheetah

R8: IF the animal is a carnivore and it is brown and has black stripes

THEN the animal is a tiger

Next, we give the rules to define ungulate category under mammal, and determine its specific animals.

R9: IF the animal is a mammal and it has hooves

THEN the animal is an ungulate

R10: IF the animal is a mammal and it chews cud

THEN the animal is a ungulate and even-toed

R11: IF the animal is a ungulate and has long neck, long legs and it is brown and has dark spots

THEN the animal is a giraffe

R12: IF the animal is a ungulate and it is white and has black stripes

THEN the animal is a zebra

Then we define three birds:

R13: IF the animal is a bird and it does not fly and has long neck, long legs and it is black and white

THEN the animal is a ostrich

R14: IF the animal is a bird and it does not fly and it swims and it is black and white

THEN the animal is a penguin

R15: IF the animal is a bird and it is a good flyer

THEN it is an albatross

The AND/OR inference net which describes this sample system is shown in Figure (Harvey, 1986). There are twelve output assertions (consequent) , rectangular nodes in the figure, and 24 input questions (antecedents). Four of these inputs serve both as assertions and questions, MAMMAL, CARNIVORE, UNGULATE and BIRD.

There are four intermediate results marked by circles which are merely connections between previous nodes and following nodes. If it is possible, these nodes should be compressed either by hand or by the optimization process of the program.

The corresponding neural network version of the sample expert system is shown in Figure 15. This network has only one layer compared to three layers in the AND/OR inference net. The four assertion/question names are implemented as feed back variables. The system will have a stable answer if the results of two runs are the same. The weights and thresholds of these T-gates were assigned as stated before. There are two mixed T-gates which should be separated into two layers (BIRD and CARNIVORE). Since only a few inputs are involved in each of these gates, we were able to combine the mixed AND/OR sub-net into one T-gate. However, this may not the case always to a two level T-gate representation has to be used. An exclusive-OR is a good example of this case.

From this neural network example, we find that more than one assertions can fire at the same time. This means that more than one results can be reached after each run. The inferencing procedure may stop at when the proper conclusion is obtained or at when two adjacent runs yield the same answer.

For instance, if the inputs facts are FEATHERS, LONG NECK, MAMMAL, HOOVES, DARK SPOTS, BROWN and LONG LEGS, then it will take two runs to fire ALBATROSS and GIRAFFE at the same time. It would need four runs to fire them in a conventional expert system which uses the inference engine designed with an AND/OR tree. If the machine which is used to implement the expert system is a sequential one, it would take much more time to reach the answers.

5. CONCLUDING REMARKS

As indicated by Fahlman and Hinton (1987), massively parallel networks of simple neuron-like processing elements may hold the key to some important aspects of intelligence not captured by existing artificial intelligence technology on serial machines. The purpose of our design is to harness these characteristics of neural networks for generating expert systems in real-time applications.

We have examined the process of constructing a neural network based expert system from a given rule-base. From the discussion presented and the results obtained from examples, it is clear that neural networks can be used as implementational vehicle for expert systems for time-critical applications. This technique will

substantially simplify expert systems and lead to a great increase in the speed performance.

For existing expert systems, the design of inference engine is a big issue which significantly affects the efficiency of the system. In all conventional expert systems, the inference engine is separated from the knowledge base. In fact, the design of the inference engine is made system independent. However, in our proposed neural network based expert systems, there are actually no designing part of the inference engine. The knowledge base and inference engine of an expert system are built into one concrete part -- the neurons and their connections. This feature is the same as that in a data flow machine. This kind of system can be considered as data-driven system. Whether a rule will fire or not depends on all input data led to this rule, and there are no other mechanism to control the firing process. While in a conventional expert system (to be considered as program-driven), the firing process of rules depend not only on their inputs but also on the inference engine which controls the firing. By eliminating the design of inference engine, the expert systems become simpler and quicker.

Some points of this design are worth to present for future study. One major difference between our design and other neural network implementation of expert systems is that we omitted the training part. In the situation where no rules are given, training the system from given data is the only way to obtain a neural network expert system. Another big advantage of using neural

networks is that it can fire rules with incomplete information. Therefore, we may add to the system a training phase which will take data to generate new rules.

The present implementation of the neural network based expert system is simulated on a conventional computer. This greatly restricts the abilities of the system. The resulting neural system is a network of connections of simple T-gates, and the weights and threshold values are fixed for a particular application. Thus, we may use VLSI technology to design the system into hardware chips. This will enormously benefiting the real-time applications. Imaging that diagnosis is done by a built-in expert system chip instead of a sizable computer for automatic machinery.

6. ACKNOWLEDGMENT

This work has been supported by a grant NAG # 3-960 from NASA Lewis Research Center, and by an UC-NASA Center grant.

REFERENCE

Aho, Alfred V., Sethi, R. and Ullman, Jeffrey D., 1986, Compilers -- Principles, Techniques, and Tools (Reading, MA.: Addison-Wesley Publishing Company).

Davis, Martin D. and Weyuker, Elaine J., 1983, Computability, Complexity and Languages (Orlando, FL.: Academic Press, Inc.).

Fahlman, Scott E. and Hinton, Geoffrey E., 1987, Connectionist

architectures for artificial intelligence. IEEE Computer, Jan., 100-109.

Forsyth, R., 1984, Expert Systems: Principles and Case Studies (New York: Chapman and Hall Company).

Gallant, Stephen I., 1988, Connectionist expert systems. Communication of the ACM, Feb., 152-169.

Harmon, P., Maus, R. and Morrissey, W., 1988, Expert Systems - Tools and Applications (New York: John Wiley & Sons, Inc.).

Harvey, J., 1986, Expert systems: An introduction. Electrical Communication, Vol. 60, No. 2, 100-108.

Hayes-Roth, F., 1985, Rule-based systems. Communication of the ACM, Sept., 921-932.

Hayes-Roth, F., Waterman, D. and Lenat, D., 1983, Building Expert Systems (New York: Addison-Wesley Publishing Company).

Heragu, Sunderesh S. and Kusiak, A., 1987, Analysis of expert systems in manufacturing design. IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-17, No. 6, Nov./Dec., 898-912.

Lippmann, Richard P., 1987, An introduction to computing with neural nets. IEEE ASSP Magazine, April, 4-22.

Winston, Patrick H., 1984, Artificial Intelligence (Reading MA.: Addison-Wesley Inc.).

Expert Systems

- Knowledge Base
- Inference Engine
- Knowledge Aquisition
- Explanatory Interface

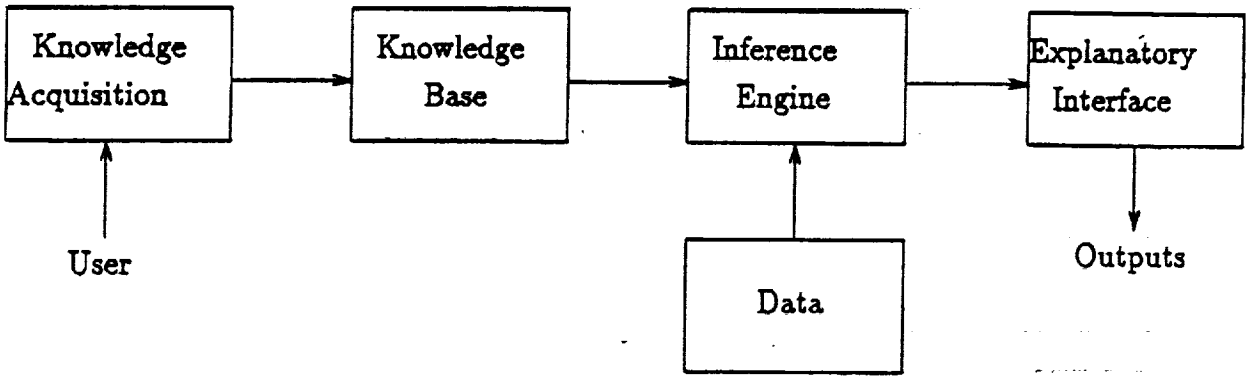

Figure 1. Block Diagram of Expert Systems

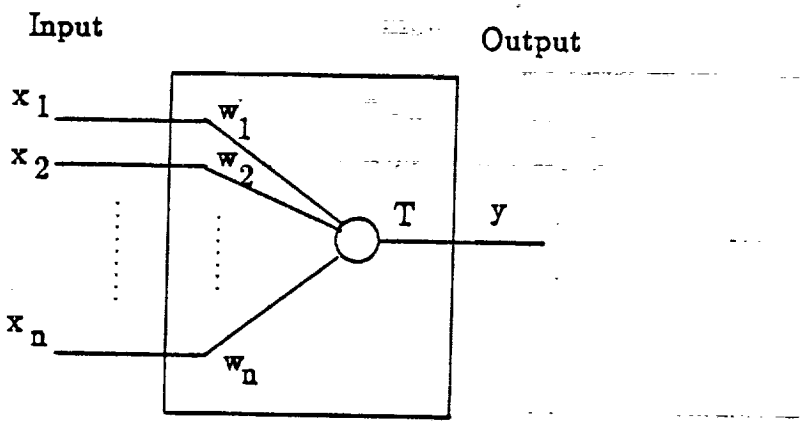

Figure 2. An Neural Network Element

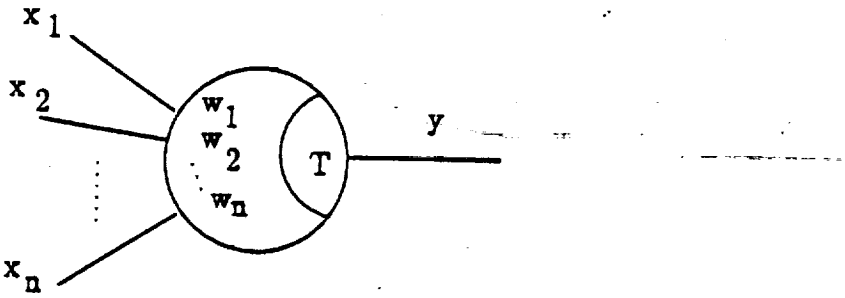

Figure 3. A Threshold Logic Gate (T-gate)

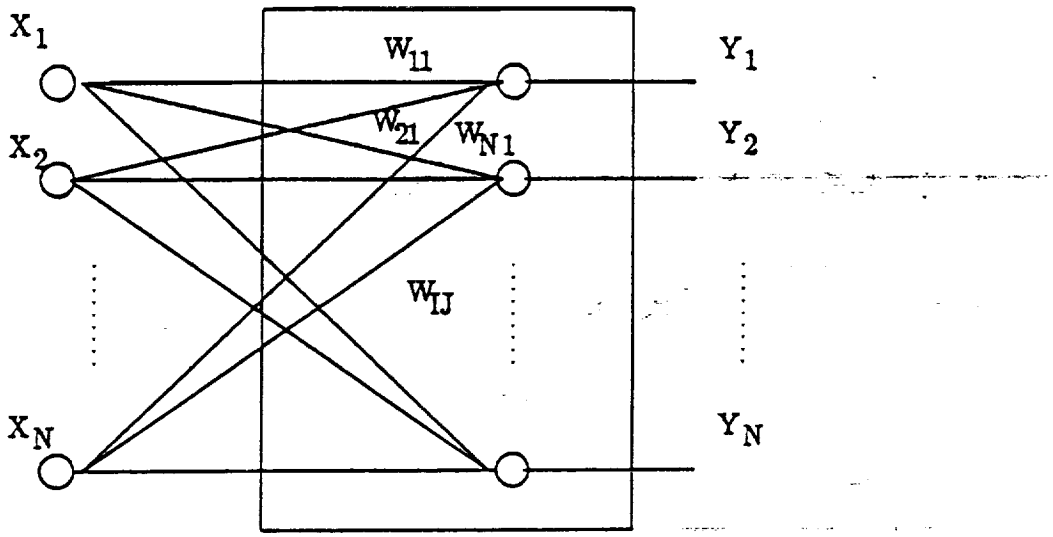

Figure 4. A Neural Network Diagram

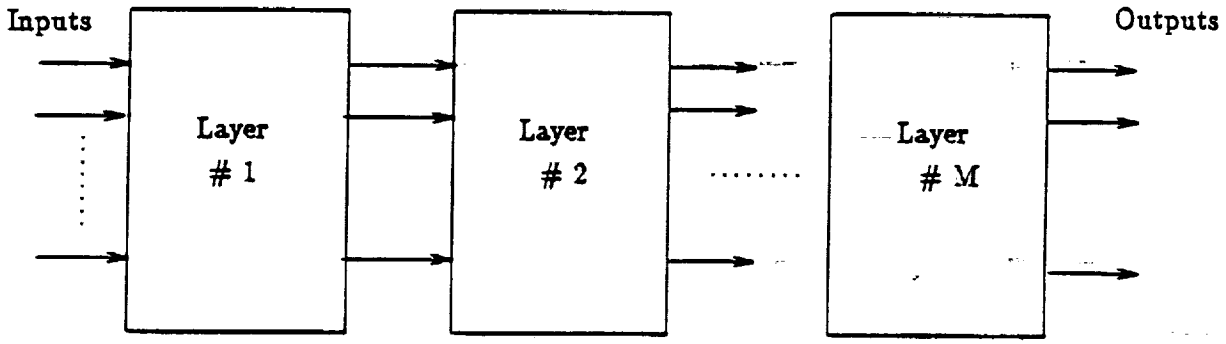

Figure 5. A multi-layer Neural Network

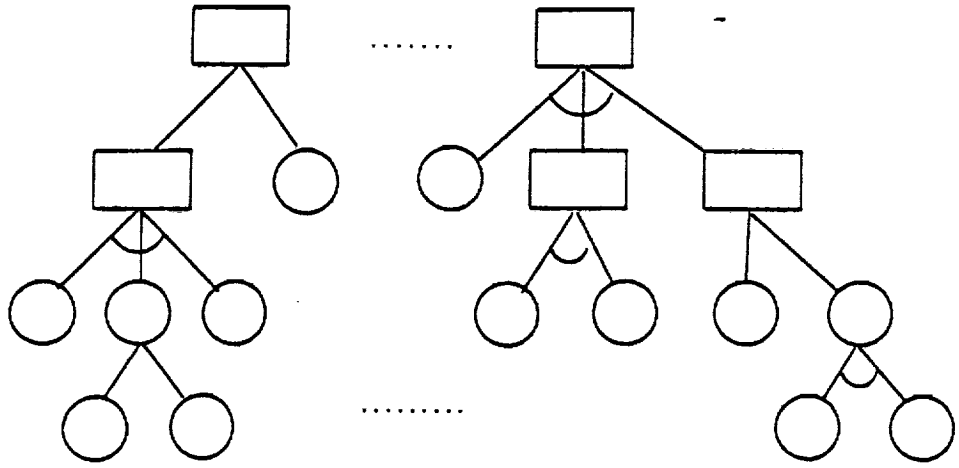

Figure 6. An AND/OR Inference Net

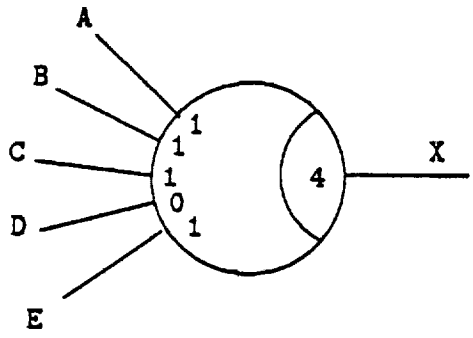

Figure 7. An AND T-gate

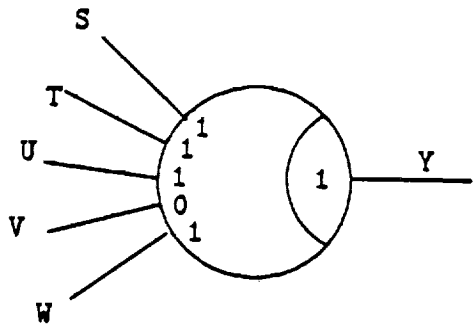

Figure 8. An OR T-gate

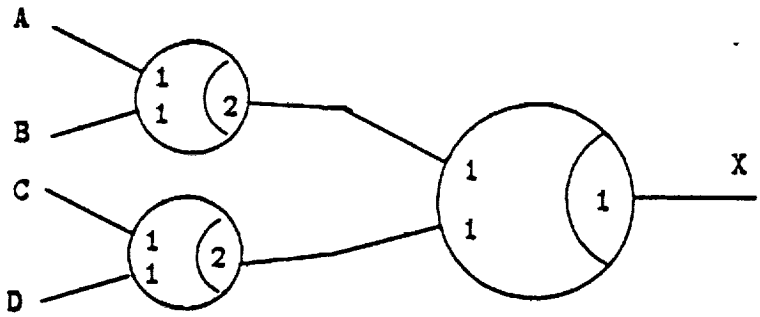

Figure 9. An Mixed AND/OR T-gate Sub-net

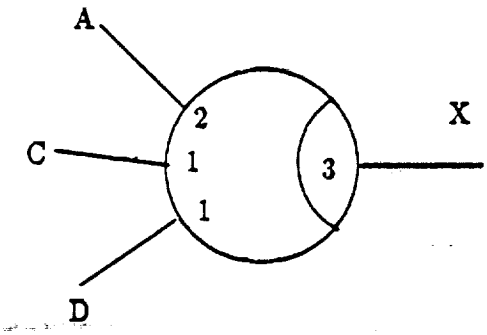

Figure 10. One T-gate for Mixed Operations

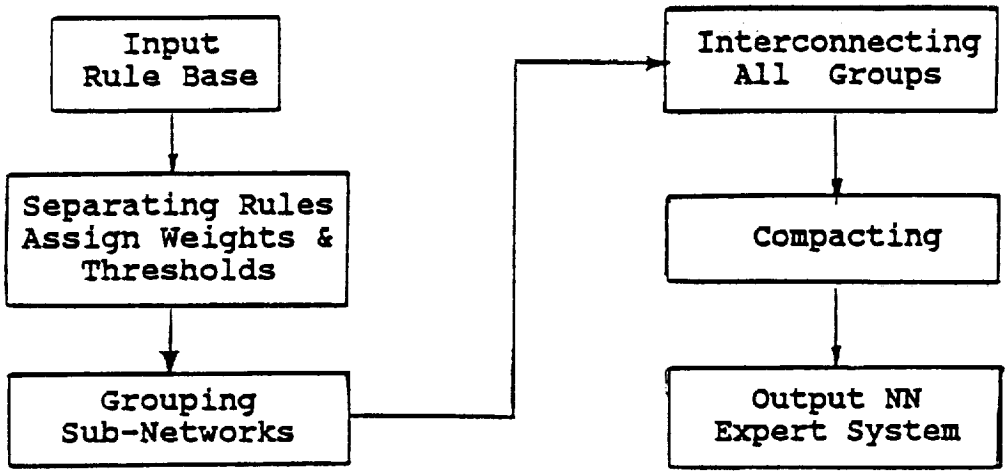

Figure 11. The Translation Steps

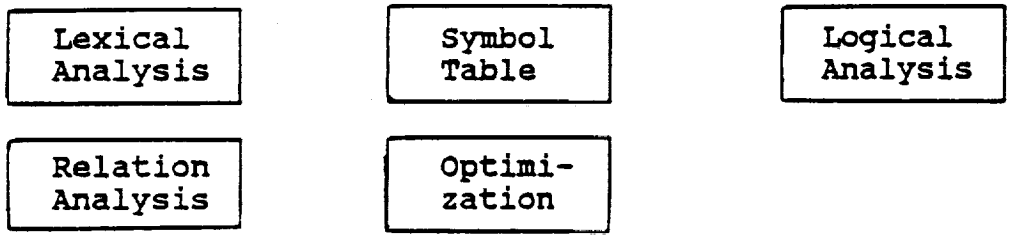

Figure 12. Design Modules for the Compiler

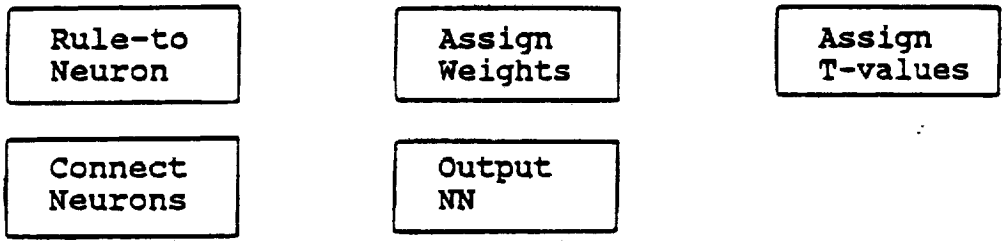

Figure 13. Design Modules for the Converter

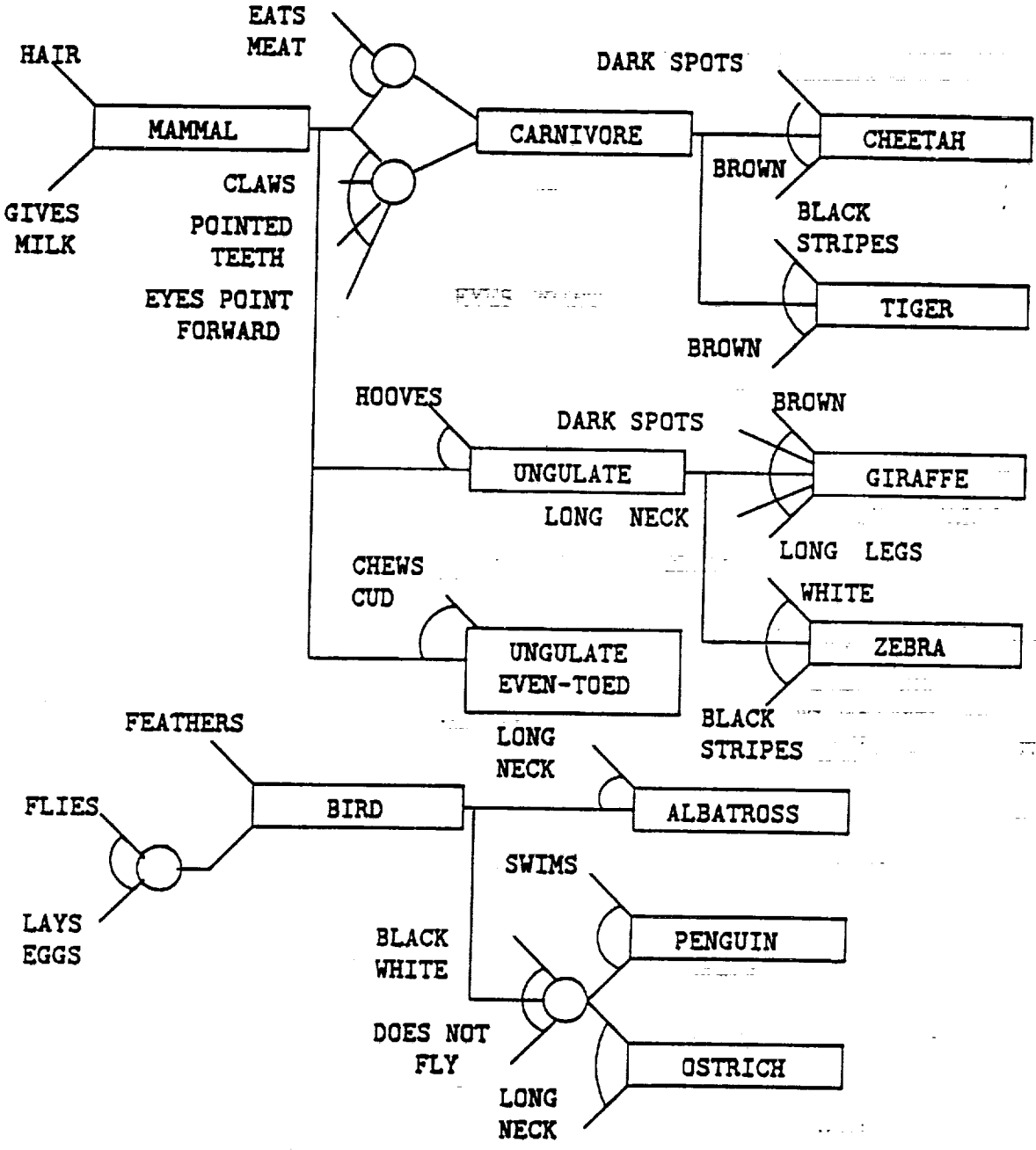

Figure 14. AND/OR Inference Net for the Sample Animal Classification System

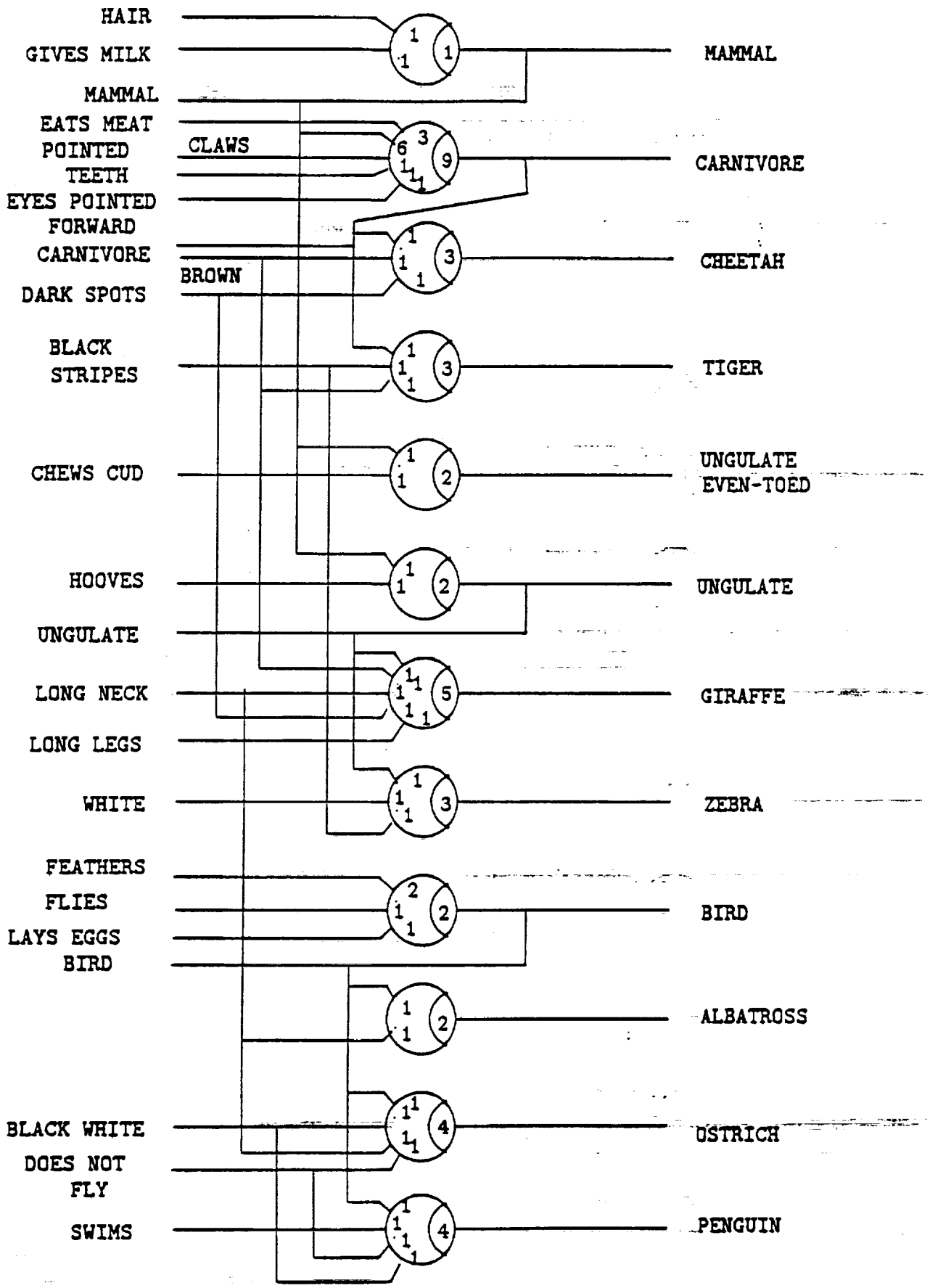

Figure 15. Neural Network for the Sample System

IEEE Intl. Conf. Systems Engg. Aug. 1991.

FUZZY EXPERT SYSTEMS VS. NEURAL NETWORKS — TRUCK BACKER-UPPER CONTROL REVISITED

P.A. Ramamoorthy and Song Huang

Department of Electrical & Computer Engineering,
University of Cincinnati, M.L. #30
Cincinnati, Ohio 45221-0030
FAX: 513-556-7326, Email: Pramamoonest.ece.uc.edu

ABSTRACT

Research on neural networks and fuzzy logic have progressed on two independent paths. In general, fuzzy logic uses verbal information for handling higher-order logical relations between inputs and outputs which are not crisply defined. On the other hand, neural networks are used to obtain information about systems from large input/output observations and training or learning procedures. From these definitions, it appears that fuzzy logic and neural network fulfill two complementary functions. Hence, a merger of these two concepts could lead to powerful yet flexible knowledge processing tools. This paper provides some insights along these lines using the truck-backer-upper control problem. New network architectures by merging these two concepts and simulation results for the truck-back-upper problem using the new architecture are also shown in this paper.

INTRODUCTION

Both neural network and fuzzy expert system are systems that map an input u (a vector of size $N \times 1$) into an output y (a vector of $M \times 1$) by the function $f : u \rightarrow y$. In a simple neural network, the mapping is performed in the system by weighing each and every inputs, summing the results, subtracting a bias value and passing the result through a non-linear function which may produce a binary or bipolar or continuous value [1,2]. Such networks may be cascaded to propagate the intermediate results to higher levels for more sophisticated problems. In the case of fuzzy expert systems, the ranges of the inputs and outputs are split into smaller and overlapping ranges or fuzzy sets. A fuzzy membership function is associated to each fuzzy subset. The mechanism governing the mapping from the input fuzzy sets to the output fuzzy sets is a collection of fuzzy rules — fuzzy rule base or fuzzy associative memories (FAM) [3,4]. The mapping from the inputs u to the output y is achieved through these fuzzy rules, the membership functions, and defuzzification procedure.

There are similarities and differences between these two mapping systems. The similarities include provision for dealing with imprecise data or data corrupted by noise, having similar primitives or building blocks to produce nonlinear mapping (membership functions, fuzzy rules, MAX-MIN or centroid operations, vs. sigmoid functions in neural networks). The major difference is that fuzzy expert systems use logic rules for inferencing while neural networks are data-driven. Therefore, fuzzy expert systems can be considered as a macroscopic tool for information processing, whereas neural networks are microscopic in nature. The advantage of neural network is their ability to learn the mapping through training. The advantages of fuzzy expert systems are their ability to provide nonlinear mapping through the membership functions and fuzzy rules, and the ability to deal with fuzzy information and incomplete and/or imprecise data. By merging the advantages of these two systems, one can arrive at a more powerful yet more flexible system for inferencing and learning. This concept will be explained through the use of results for the truck-backer-upper control problem.

PROBLEM DEFINITION

The truck backer-upper control is a typical nonlinear control problem where a controller to successfully back up a truck to a loading dock from any reasonable initial location has to be designed. Nguyen and Widrow [5] showed that a nonlinear controller using a two layer neural network architecture with 26 adaptive neural elements can be successfully trained. Recently, Kong and Kosko [6] compared the performance of such a neural network based controller with that of a controller based on fuzzy expert system composed of 35 rules. They observed that even that simple fuzzy expert system lead to smoother trajectories than that produced by the two-layer neural network. If their observations are valid in general, it is desirable to arrive at a logical explanation for the differences in the performances. More importantly, as stated earlier, approaches that can retain the attractive properties of neural networks and at the same time obtain performances comparable to that of fuzzy expert systems need to be developed.

Figure 1 shows the loading zone of the truck-backer-upper problem and inputs and output variables of the system. If enough clearance is given between the truck and the loading dock, then the y-position can be omitted as a input to tune the controller. The ranges of the inputs, x-position and the truck orientation angle θ, and the output, steering signal ϕ, are given as:

$$\phi: [\,0,\,360|;\ x: [\,0,\,100|;\ \theta: (\,-30,\,30|$$

Having identified the variables and their ranges, fuzzy sub-